Computer Programming: Skills & Concepts (CP1)
Structured data: arrays

19th October, 2010

Motivation for arrays

In our program on “coin changing” we introduced individual integer variables to keep track of the number of coins of each denomination:

```
int n1, n2, n3, n4, n5, n6, n7, n8;
```

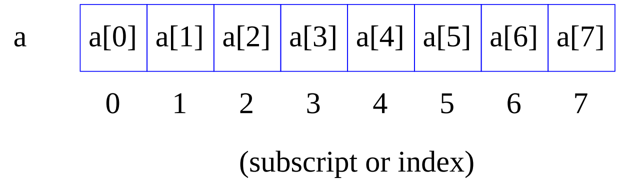
When it came to updating these variables we had to resort to a lengthy conditional statement, with a separate case for each of the seven variables. There ought to be a better way!

Declaration of arrays

The declaration

```
#define SIZE 8
int a[SIZE];
```

introduces an *array*, called *a*, with 8 *elements* (or *components*) of type integer.



a	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
	0	1	2	3	4	5	6	7

(subscript or index)

Notes

- The first element of the array has index 0, and the final element has index `SIZE - 1`.
- We refer to the entire array as `a`.
- All the elements of the array have type `int`. We refer to these individual elements as `a[0]`, `a[1]`, and so on up to `a[SIZE - 1]`.
- Array indices are expressions of type `int`

Where the power lies

Since an array index is a integer *expression*, and not a *constant*, its value isn't determined until the program is run. The precise array element referred to by a[i] depends on the current value of i
Example:

```
for (i = 0;  i < SIZE;  ++i) a[i] = 0;
```

Effect: Initialise all elements of the array a to zero.
C.f.

```
a[0] = 0;
a[1] = 0;
...
a[SIZE - 1] = 0;
```

Letter frequencies with arrays

```
int c,  i, count[26];  /* Allocate one counter per letter */

for (i = 0; i <= 25; ++i) count[i] = 0;
while ((c = getchar()) != EOF) {
  c = toupper(c);
  if (isupper(c)) {
    i = c - ’A’;      /* Integer in [0,25] */
    ++count[i];       /* Increment counter for letter just read */
  }
}
for (i = 0; i <= 25; ++i)
  printf("%c: %d\n", i + ’A’, count[i]);  /* Print frequencies */
```

Finding the commonest letter

```
int maxCount,   /* Maximum count seen so far */
    maxIndex;   /* Location where we observed that maximum */

maxCount = count[0];   /* Letter A is deemed the winner, */
maxIndex = 0;          /* at the outset.                 */
for (i = 1;  i <= 25;  ++i) {
  if (count[i] > maxCount) {   /* Bigger than seen so far? */
    maxCount = count[i];
    maxIndex = i;
  }
}
printf("The commonest letter is \"%c\" with %d occurrences.",
  ’A’ + maxIndex, maxCount);
```

Arrays of any type

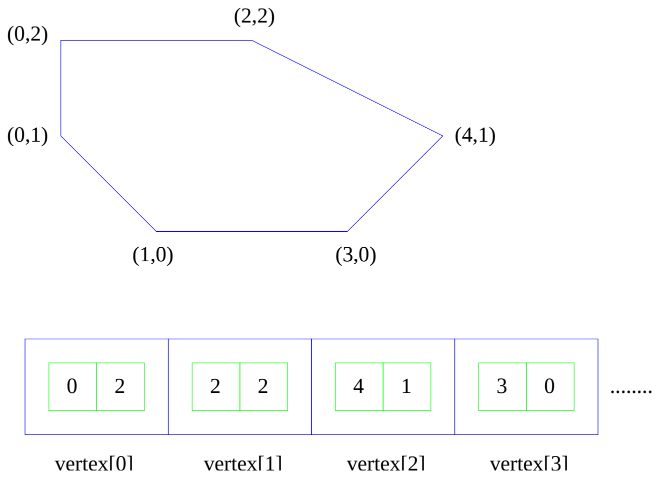
We haven't discussed typedef or struct formally yet ... though we have seen, in Practical 1, their use to define a type for representing points in the plane.
An array of points could be used to represent a polygon with up to MAX vertices.

```
typedef struct {
  int x, y;
} point_t;

point_t vertex[MAX];
```

Question: How do we deal with a polygon with fewer than MAX vertices?

Polygon as an array of vertices



Arrays as parameters

```
int Max(int a[], int n) {
/*  n is the number of elements in array a.  Max returns
 *  the maximum element of a.  NB:  We lose the size of
 *  the array when we pass it as a parameter             */

  int i, maxSoFar;
  maxSoFar = a[0];
  for (i = 1;  i < n;  ++i)
    if (a[i] > maxSoFar) maxSoFar = a[i];
  return maxSoFar;
}

printf("The commonest letter occurred %d times.", Max(count, 26));
```

Arrays are "pointers"

```
void Rotate(int a[], int n) {
/* Aim:  rotate the elements of a cyclically one position. */
  int i;
  int temp;  /* Temporary storage location (like in swap). */

  temp = a[n - 1];
  for (i = n - 1;  i > 0;  --i) a[i] = a[i - 1];
  a[0] = temp;
}

Rotate(count, 26);
```

Question: Is `count` cyclically rotated or unchanged?

Arrays are "pointers"

The answer is that it *is* rotated.
The reason? Roughly it is because an array in C is a pointer (to its first element).

- The actual parameter `count` is a pointer to an integer.
- The formal parameter `a[]` is a synonym for `*a`.

+ve: Means we don't need to use & and * to get the effect of "call-by-reference" with array parameters. (remember swap from lecture 9).

-ve: We always have to incorporate an extra parameter (eg, `n` in `Rotate`) to allow the length of the array to be passed into the function.

Arrays of arrays

Array elements can themselves be arrays. So, for example, a matrix with `N` rows and `M` columns could be defined as:

```
float matrix[N][M];
```

We'd then expect to be able to write a function that multiplies a vector `x` by a matrix `a` with header

```
void LinTransform(float a[][],
                  float x[],
                  float y[],
                  int n, int m);
```

However C does *not* allow this - declaration for `a` must instead be of the form `a[][10]` or `a[][8]` or similar.
To understand why, check out Kelley & Pohl [KP, §6.12].

Coin Changing with arrays

Use an array to store the counts `n1, ..., n8` in a common format.

- Don't need global variables any more

Reading Material

Most of Chapter 6, Kelley and Pohl.

- Specifically, 6.1, 6.4, 6.6, 6.12
- Some other sections of chapter 6 discuss pointers, and also the relationship between pointers and arrays.